COP 4516: Competitive Programming and Problem Solving

Giri Narasimhan & Kip Irvine
Phone: x3748 & x1528
{giri,irvinek}@cs.fiu.edu
Evaluation

- Exam/Competition: 50%
- Solving Problems: 40%
- Attendance: 5%
- Class Participation: 5%
The great thinkers of our field:

- **Euclid**, 300 BC
- **Bhaskara**, 6\(^{th}\) century
- **Al Khwarizmi**, 9th century
- **Fibonacci**, 13\(^{th}\) century
- **Babbage**, 19\(^{th}\) century
- **Turing**, 20\(^{th}\) century
- **von Neumann, Knuth, Karp, Tarjan**, …
Al Khwarizmi’s algorithm

- **43 X 17**
  - 43 17
  - 21 34
  - 10 68 (ignore)
  - 5 136
  - 2 272 (ignore)
  - 1 544

-------------------

731
Euclid’s Algorithm

- $\text{GCD}(12,8) = 4$; $\text{GCD}(49,35) = 7$;
- $\text{GCD}(210,588) = ??$
- $\text{GCD}(a,b) = ??$

- **Observation**: [a and b are integers and $a \geq b$]
  - $\text{GCD}(a,b) = \text{GCD}(a-b,b)$

- **Euclid’s Rule**: [a and b are integers and $a \geq b$]
  - $\text{GCD}(a,b) = \text{GCD}(a \mod b, b)$

- **Euclid’s GCD Algorithm**:
  - $\text{GCD}(a,b)$
    - If $(b = 0)$ then return $a$;
    - return $\text{GCD}(a \mod b, b)$
If you like Algorithms, nothing to worry about!

"Calculus is my new Versace. I get a buzz from algorithms. What's going on with me, Raymond? I'm scared."
Search

• You are asked to guess a number $X$ that is known to be an integer lying in the range $A$ through $B$. How many guesses do you need in the worst case?
  - Use binary search; Number of guesses = $\log_2(B-A)$

• You are asked to guess a positive integer $X$. How many guesses do you need in the worst case?
  - **NOTE**: No upper bound is known for the number.
  - **Algorithm**:
    - figure out $B$ (by using **Doubling Search**)
    - perform binary search in the range $B/2$ through $B$.
  - Number of guesses = $\log_2B + \log_2(B - B/2)$
  - Since $X$ is between $B/2$ and $B$, we have: $\log_2(B/2) < \log_2X$,
  - Number of guesses < $2\log_2X - 1$
Polynomial Evaluation

• **Given a polynomial**
  
  \[ p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n \]

  compute the value of the polynomial for a given value of \( x \).

• **How many additions and multiplications are needed?**
  
  – **Simple solution:**
    • Number of additions = \( n \)
    • Number of multiplications = \( 1 + 2 + \ldots + n = \frac{n(n+1)}{2} \)

  – **Reusing previous computations:** \( n \) additions and \( 2n \) multiplications!

  – **Improved solution using Horner’s rule:**
    • \( p(x) = a_0 + x(a_1 + x(a_2 + \ldots x(a_{n-1} + x a_n)) \ldots)) \)
    • Number of additions = \( n \)
    • Number of multiplications = \( n \)
Sorting

• Input is a list of $n$ items that can be compared.
• Output is an ordered list of those $n$ items.
• Fundamental problem that has received a lot of attention over the years.
• Used in many applications.
• Scores of different algorithms exist.
• Task: To compare algorithms
  - On what bases?
    • Time
    • Space
    • Other
Figure 2.1  Sorting a hand of cards using insertion sort.
Sorting Algorithms

- Number of Comparisons
- Number of Data Movements
- Additional Space Requirements
Sorting Algorithms

- SelectionSort
- InsertionSort
- BubbleSort
- ShakerSort
- MergeSort
- HeapSort
- QuickSort
- Bucket & Radix Sort
- Counting Sort
SelectionSort

SelectionSort(array A)

1. $N \leftarrow \text{length}[A]$
2. for $p \leftarrow 1$ to $N$
3. do Compute $j$, the index of the smallest item in $A[p..N]$
SelectionSort

\textbf{SelectionSort}(array } A \textbf{)}

1. \( N \leftarrow \text{length}[A] \)
2. \textbf{for } p \leftarrow 1 \textbf{ to } N \textbf{ do} \triangleright \text{Compute } j
3. \hspace{1em} j \leftarrow p
4. \hspace{1em} \textbf{for } m \leftarrow p + 1 \textbf{ to } N \textbf{ do if } (A[m] < A[j])
5. \hspace{2em} \textbf{then } j \leftarrow m \triangleright \text{Swap } A[p] \text{ and } A[j]
6. \textbf{end do}
7. \hspace{1em} temp \leftarrow A[p]
8. \hspace{1em} A[p] \leftarrow A[j]
9. \hspace{1em} A[j] \leftarrow temp
SelectionSort

SelectionSort(array A)

1 $N \leftarrow \text{length}[A]$
2 $\textbf{for } p \leftarrow 1 \textbf{ to } N \textbf{ do } \triangleright \text{Compute } j$
3 \hspace{1em} $j \leftarrow p$
4 $\textbf{for } m \leftarrow p + 1 \textbf{ to } N \textbf{ do if } (A[m] < A[j])$
5 \hspace{1em} $\textbf{then } j \leftarrow m$
6 \hspace{1em} $\triangleright \text{Swap } A[p] \text{ and } A[j]$
7 $\text{temp} \leftarrow A[p]$
8 $A[p] \leftarrow A[j]$
9 $A[j] \leftarrow \text{temp}$

O(n²) time
O(1) space
Solving Recurrence Relations

Page 62, [CLR]

<table>
<thead>
<tr>
<th>Recurrence; Cond</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = T(n-1) + O(1) )</td>
<td>( T(n) = O(n) )</td>
</tr>
<tr>
<td>( T(n) = T(n-1) + O(n) )</td>
<td>( T(n) = O(n^2) )</td>
</tr>
<tr>
<td>( T(n) = T(n-c) + O(1) )</td>
<td>( T(n) = O(n) )</td>
</tr>
<tr>
<td>( T(n) = T(n-c) + O(n) )</td>
<td>( T(n) = O(n^2) )</td>
</tr>
<tr>
<td>( T(n) = 2T(n/2) + O(n) )</td>
<td>( T(n) = O(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = aT(n/b) + O(n); \quad a = b )</td>
<td>( T(n) = O(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = aT(n/b) + O(n); \quad a &lt; b )</td>
<td>( T(n) = O(n) )</td>
</tr>
<tr>
<td>( T(n) = aT(n/b) + f(n); \quad f(n) = O(n^{\log_b a-\epsilon}) )</td>
<td>( T(n) = O(n) )</td>
</tr>
<tr>
<td>( T(n) = aT(n/b) + f(n); \quad f(n) = O(n^{\log_b a}) )</td>
<td>( T(n) = \Theta(n^{\log_b a \log n}) )</td>
</tr>
<tr>
<td>( T(n) = aT(n/b) + f(n); \quad f(n) = \Theta(f(n)); \quad af(n/b) \leq cf(n) )</td>
<td>( T(n) = \Omega(n^{\log_b a \log n}) )</td>
</tr>
</tbody>
</table>
**Insertion-Sort**($A$)

1. **for** $j \leftarrow 2$ **to** $\text{length}[A]$  
2. **do** $key \leftarrow A[j]$  
4. $i \leftarrow j - 1$  
5. **while** $i > 0$ and $A[i] > key$  
6. **do** $A[i + 1] \leftarrow A[i]$  
7. $i \leftarrow i - 1$  
8. $A[i + 1] \leftarrow key$

Loop invariants and the correctness of insertion sort
**Insertion-Sort(A)**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Cost and Times**

<table>
<thead>
<tr>
<th></th>
<th>cost</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₁</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>c₂</td>
<td>n−1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>n−1</td>
<td></td>
</tr>
<tr>
<td>c₄</td>
<td>n−1</td>
<td></td>
</tr>
<tr>
<td>c₅</td>
<td>(\sum_{j=2}^{n} t_j)</td>
<td></td>
</tr>
<tr>
<td>c₆</td>
<td>(\sum_{j=2}^{n} (t_j - 1))</td>
<td></td>
</tr>
<tr>
<td>c₇</td>
<td>(\sum_{j=2}^{n} (t_j - 1))</td>
<td></td>
</tr>
<tr>
<td>c₈</td>
<td>n−1</td>
<td></td>
</tr>
</tbody>
</table>

**O(n²) time**

**O(1) space**
Figure 2.4 The operation of merge sort on the array $A = (5, 2, 4, 7, 1, 3, 2, 6)$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.
Figure 2.3  The operation of lines 10–17 in the call **MERGE**(A, 9, 12, 16), when the subarray
A[9…16] contains the sequence (2, 4, 5, 7, 1, 2, 3, 6). After copying and inserting sentinels, the
array \( L \) contains (2, 4, 5, 7, \( \infty \)), and the array \( R \) contains (1, 2, 3, 6, \( \infty \)). Lightly shaded positions
in \( A \) contain their final values, and lightly shaded positions in \( L \) and \( R \) contain values that have yet
to be copied back into \( A \). Taken together, the lightly shaded positions always comprise the values
originally in \( A[9…16] \), along with the two sentinels. Heavily shaded positions in \( A \) contain values
that will be copied over, and heavily shaded positions in \( L \) and \( R \) contain values that have already
been copied back into \( A \). (a)–(h) The arrays \( A, L, \) and \( R, \) and their respective indices \( k, i, \) and \( j \)
prior to each iteration of the loop of lines 12–17. (i) The arrays and indices at termination. At this
point, the subarray in \( A[9…16] \) is sorted, and the two sentinels in \( L \) and \( R \) are the only two elements
in these arrays that have not been copied into \( A \).
Assumption: Array $A$ is sorted from positions $p$ to $q$ and also from positions $q+1$ to $r$. 

```
MERGE(A, p, q, r)
1    n_1 ← q - p + 1
2    n_2 ← r - q
3    create arrays $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$
4    for $i ← 1$ to $n_1$
5        do $L[i] ← A[p + i - 1]$
6    for $j ← 1$ to $n_2$
7        do $R[j] ← A[q + j]$
8    $L[n_1 + 1] ← ∞$
9    $R[n_2 + 1] ← ∞$
10   $i ← 1$
11   $j ← 1$
12   for $k ← p$ to $r$
13       do if $L[i] ≤ R[j]$
14          then $A[k] ← L[i]$
15          $i ← i + 1$
16       else $A[k] ← R[j]$
17          $j ← j + 1$
```
\textbf{MERGE-SORT}(A, p, r)

1 \textbf{if} \, p < r

2 \textbf{then} \, q \leftarrow \lfloor (p + r)/2 \rfloor

3 \textbf{MERGE-SORT}(A, p, q)

4 \textbf{MERGE-SORT}(A, q + 1, r)

5 \textbf{MERGE}(A, p, q, r)
Figure 2.5 The construction of a recursion tree for the recurrence $T(n) = 2T(n/2) + cn$. Part (a) shows $T(n)$, which is progressively expanded in (b)–(d) to form the recursion tree. The fully expanded tree in part (d) has $\log{n} + 1$ levels (i.e., it has height $\log{n}$, as indicated), and each level contributes a total cost of $cn$. The total cost is $cn \log{n} + cn$, which is $\Theta(n \log{n})$. 